Compiling *FORTRAN* and *C* Programs on *UNIX* Machines

There are basically two ways of compiling and linking *FORTRAN* and *C* code:
A. Using commands which call *FORTRAN* or *C* compilers directly
Compilers under *UNIX* will by default not only compile but also create an executable module (compile and link). Pure compilation requires the option -c.
B. Using the make command and makefiles
The make command is used to compile/link or recompile/relink programs/modules that have been changed since the last compilation, or that are affected by changes of other modules.

A. Using commands which call *FORTRAN* or *C* compilers directly

In the case of strange behaviour of your *FORTRAN* programs, first try to switch off the optimization (-O0) or switch on the debug option (-g). Then you can use nice interactive debuggers such as **cvd** or **dbx**. Please note that the option -g will expands your libraries and executable modules considerably.

a) Silicon Graphics - SGI Challenge machines (x4u1, x4u2, dice1, dice2, zarah1, zarah2, hermes)

```
FORTRAN :  f77     [ options | file ]
           Recommended for production :  f77  test.prog.f  -static  -G 0  -mips2  -O  -o test.prog
           Recommended for testing :     f77  test.prog.f  -static  -G 0  -mips2  -g   -o test.prog
C :        cc      [ options | file ]
           Recommended for production :  cc  test.prog.c  -G 0  -mips2  -cckr  -O  -o test.prog
           Recommended for testing :     cc  test.prog.c  -G 0  -mips2  -cckr  -g   -o test.prog
```

For the old SGI Power Series machines (x4u, rec01 to rec06) you must not use the option *–mips2*.

b) Hewlett Packard - HP (ips102 to ips119)

```
FORTRAN :  f77     [ options | file ] ... doesn't allow -L option
           fort77  [ options | file ] ... allows -L option
           Recommended for production :  fort77  test.prog.f  -K  +ppu  -O  -o test.prog
           Recommended for testing :     fort77  test.prog.f  -K  +ppu  -g   -o test.prog
C :        cc      [ options | file ] ... HP-UX compiler
           c89     [ options | file ] ... HP-UX POSIX compiler
           Recommended for production :  cc  test.prog.c  -O  -o test.prog
           Recommended for testing :     cc  test.prog.c  -g   -o test.prog
```

Note: –**O** means optimization, –**g** produces symbol tables for debugging, –**static** (on SGI) or –**K** (on HP) save local FORTRAN variables as on the IBM, –**mips2** (on SGI) produces native code for the Challenge machines, –**cckr** (on SGI) accepts the K&R variant of C, **+ppu** (on HP) appends underscores to external symbols.

B.1. The make command

The **make** command searches the current directory for a file with the name **makefile, Makefile, s.makefile**, or **s.Makefile**, in that order. The **-f** option may be used to override this naming convention (i.e. **make -f other.makefile.name**).
The make command reads the makefile for information about the specified TargetFiles and for the commands necessary for updating them. The make command changes the TargetFile only if you have changed one or more of the source files. It considers a missing file to be a changed file (out-of-date).
To force compilation under all circumstances use **make -u**.

B.2. Format of makefiles

The general form of an entry is:

```
Target1  [ Target2 ... ]  : [:]  [ Parent1 ... ] [ ;command ] ...
[(tab)   commands ]
```

Items inside brackets are optional.
Each line in the makefile that contains a target file name is called a **dependency line**.
Lines that contain only commands must begin with a **tab** character.

The **make** program ignores blank lines and **comment lines**, which begin with a # (pound sign).
Except for comment lines, you can enter lines longer than the line width of the input device. **Lines can be continued** by placing a \ (backslash) at the end of it.

The **make** command is very sensitive to any missing tab or additional blanks, e.g. after a backslash. If your make command complaints about missing libraries etc., please check your makefile in this respect.

The most frequently used form of the make command

a) **make**
Example of a makefile:

```
# Compile and link the file test.prog.c
test.prog: test.prog.c
(tab)    cc  test.prog.c  -o test.prog
```

Explanation: If test.prog.c has been modified/created, the command cc will be executed. The executable module will be called test.prog.

b) **make test.prog**
Example of a makefile:

```
# Compile and link all the FORTRAN or C programs specified as a parameter in the make command
.f:
(tab)    f77  $*.f  -o $*
.c:
(tab)    cc  $*.c  -o $*
```

Explanation: The $* (an internal macro, see below) will be replaced by the input parameter test.prog. If test.prog.f exists and has been changed since the executable test.prog has been created, the command f77 will be executed. If test.prog.c exists and has been changed since the executable test.prog has been created, the command cc will be executed. The executable module will be called test.prog.

c) **make abxy**
Example of a makefile:

```
# Compile and link the file test.prog.f
abxy: test.prog.f
(tab)    f77  test.prog.f  -o test.prog
```

Explanation: If test.prog.f has been changed more recently then the executable test.prog, the command f77 will be executed. The executable module will be called test.prog.

B.3. Macros in makefiles

Entries of the form String1=String2 are **macro definitions**.
String2 extends to the end of the line (including continuations), or to the comment character (#). Each $(String1) occuring in the makefile after the macro definition will be replaced by String2. Nested macros of the form of $($(String1)) are also allowed. The parentheses are necessary for macro names longer than one character.
The make program has some built-in internal macro definitions. The most frequently used internal macro is $*, which returns the source file name without the suffix.
The $$ symbol must be used if you need a dollar sign in a makefile. A single $ is interpreted by the make command as a macro to be evaluated, as in the above definitions.
The command **make -p** displays the complete set of macro definitions and target descriptions before performing any commands.

Example of a makefile for H1 or ZEUS

```
F77OPTIONS='list_of_FORTRAN_options'
.f:
(tab)    f77  $*.f  $(F77OPTIONS)  -o $* \
(tab)    -L /h1/lib -lh1util -lbos -lfpack \
(tab)    -L /cern/pro/lib -lpacklib
```

```
F77OPTIONS='list_of_FORTRAN_options'
.f:
(tab)    f77  $*.f  $(F77OPTIONS)  -o $* \
(tab)    -L /zeus/pro/lib -lzrecon -lphantom \
(tab)    -L /cern/pro/lib -lpacklib
```